DEPARTMENT OF COMPUTER SCIENCE COURSE SYLLABUS

CS 46101/56101  DESIGN AND ANALYSIS OF ALGORITHMS  3 credit hours

Instructor’s Name: Feodor F. Dragan


Course Catalogue:
(Cross-listed with CS 56101) Introduction to algorithmic concepts, design and complexity analysis of algorithms and data structures, searching, sorting, graph and string algorithms.
Prerequisites or co-requisites: MATH 12003 and C (2.000) or better in CS 23001
Required (BS), elective (MS, MA)

Course Goals:
1. Study important advanced data structures and algorithmic techniques not normally covered in CS 33001.
2. Develop ability to design efficient algorithms.
3. Develop ability to prove correctness and evaluate efficiency of algorithms.
4. The main goal of the course is to learn to think algorithmically like a “real” computer scientist.

Topics to be covered:
1. Basic Analysis:
   - Differences among best, expected, and worst case behaviors of an algorithm
   - Asymptotic analysis of upper and expected complexity bounds
   - Big O notation: formal definition
   - Complexity classes, such as constant, logarithmic, linear, quadratic, and exponential
   - Time and space trade-offs in algorithms
   - Big O notation: use
   - Little o, big omega and big theta notation
   - Recurrence relations
   - Analysis of iterative and recursive algorithms
   - Some version of a Master Theorem
2. Algorithmic Strategies
   - Brute-force algorithms
   - Greedy algorithms
   - Divide-and-conquer
   - Dynamic Programming
3. Fundamental Data Structures and Algorithms
Simple numerical algorithms, such as computing the average of a list of numbers, finding the min, max, and mode in a list
- Sequential and binary search algorithms
- Worst case quadratic sorting algorithms (selection, insertion)
- Worst or average case O(N log N) sorting algorithms (quicksort, heapsort, merge-sort)
- Hash tables, including strategies for avoiding and resolving collisions
- Binary search trees
  - Common operations on binary search trees such as select min, max, insert, delete, iterate over tree
- Trees, Properties, Traversal strategies
- Undirected, Directed graphs, Weighted graphs, Spanning trees
- Graphs and graph algorithms
  - Representations of graphs (e.g., adjacency list, adjacency matrix)
  - Depth- and breadth-first traversals
- Heaps
- Graphs and graph algorithms
  - Shortest-path algorithms (Dijkstra’s and Floyd’s algorithms)
  - Minimum spanning tree (Prim’s and Kruskal’s algorithms)
- Pattern matching and string/text algorithms (e.g., substring matching, regular expression matching, longest common subsequence algorithms)

4. Advanced Data Structures and Algorithms
- Balanced trees (e.g., red-black trees)
- Graphs (e.g., topological sort, finding strongly connected components, matching)

**Learning Outcomes:**

1. **Basic Analysis:**
   - Explain what is meant by “best”, “expected”, and “worst” case behavior of an algorithm. [Familiarity]
   - In the context of specific algorithms, identify the characteristics of data and/or other conditions or assumptions that lead to different behaviors. [Assessment]
   - Determine informally the time and space complexity of simple algorithms. [Usage]
   - State the formal definition of big O. [Familiarity]
   - List and contrast standard complexity classes. [Familiarity]
   - Give examples that illustrate time-space trade-offs of algorithms. [Familiarity]
   - Use big O notation formally to give asymptotic upper bounds on time and space complexity of algorithms. [Usage]
   - Explain the use of big omega, big theta, and little o notation to describe the amount of work done by an algorithm. [Familiarity]
   - Use recurrence relations to determine the time complexity of recursively defined algorithms. [Usage]
   - Solve elementary recurrence relations, e.g., using some form of a Master Theorem. [Usage]
2. **Algorithmic Strategies**
   - For each of the strategies (brute-force, greedy, divide-and-conquer, and dynamic programming), identify a practical example to which it would apply. [Familiarity]
   - Use a greedy approach to solve an appropriate problem and determine if the greedy rule chosen leads to an optimal solution. [Assessment]
   - Use a divide-and-conquer algorithm to solve an appropriate problem. [Usage]
   - Use dynamic programming to solve an appropriate problem. [Usage]
   - Determine an appropriate algorithmic approach to a problem. [Assessment]

3. **Fundamental Data Structures and Algorithms**
   - Implement basic numerical algorithms. [Usage]
   - Implement simple search algorithms and explain the differences in their time complexities. [Assessment]
   - Be able to implement common quadratic and O(N log N) sorting algorithms. [Usage]
   - Describe the implementation of hash tables, including collision avoidance and resolution. [Familiarity]
   - Discuss the runtime and memory efficiency of principal algorithms for sorting, searching, and hashing. [Familiarity]
   - Discuss factors other than computational efficiency that influence the choice of algorithms, such as programming time, maintainability, and the use of application-specific patterns in the input data. [Familiarity]
   - Explain how tree balance affects the efficiency of various binary search tree operations. [Familiarity]
   - Illustrate by example the basic terminology of graph theory, as well as some of the properties and special cases of each type of graph/tree. [Familiarity]
   - Demonstrate different traversal methods for trees and graphs, including pre-, post-, and in-order traversal of trees. [Usage]
   - Model a variety of real-world problems in computer science using appropriate forms of graphs and trees, such as representing a network topology or the organization of a hierarchical file system. [Usage]
   - Show how concepts from graphs and trees appear in data structures, algorithms, proof techniques (structural induction), and counting. [Usage]
   - Explain how to construct a spanning tree of a graph. [Usage]
   - Solve problems using fundamental graph algorithms, including depth-first and breadth-first search. [Usage]
   - Demonstrate the ability to evaluate algorithms, to select from a range of possible options, to provide justification for that selection, and to implement the algorithm in a particular context. [Assessment]
   - Describe the heap property and the use of heaps as an implementation of priority queues. [Familiarity]
   - Solve problems using graph algorithms, including single-source and all-pairs shortest paths, and at least one minimum spanning tree algorithm. [Usage]
   - Trace and/or implement a string-matching algorithm. [Usage]

4. **Advanced Data Structures and Algorithms**
o Explain how tree balance affects the efficiency of various binary search tree operations. [Familiarity]
o Solve problems using graph algorithms, including single-source and all-pairs shortest paths, at least one minimum spanning tree algorithm, topological sort, finding strongly connected components, matching. [Usage]

**Learning Outcomes Assessment:**
1. (Bi-)Weekly homework assignments on designing and analysis of algorithms;
2. Midterm Exam
3. Final Exam